

---

# IZVEŠTAJ ZA PROJEKAT BROJ 10 IZ RAČUNARSKE ELEKTRONIKE

---

Šifra predmeta: 13E043RE

radili:

Gvozdenović Ivan 0126/2014

Vuković Aleksandar 0140/2014

---

# SADRŽAJ IZVEŠTAJA:

---

postavka zadatka: .....	3
Projekat 10 - dešifrovanje upotrebom Trifid algoritma.....	3
Koncept: .....	4
Main procedura .....	4
Encryption procedura.....	4
Decryption procedura .....	5
Test Primeri: .....	5
Interfejs u komandnoj liniji:.....	6
Napomene:.....	8
Kod programa:.....	8

---

# POSTAVKA ZADATKA:

---

## Projekat 10 - dešifrovanje upotrebom Trifid algoritma

Napisati program koji učitava ulazni fajl koji u prvom redu sadrži karakter mode in{e, d} koji definiše da li tekst koji se nalazi u fajlu treba šifrovati (e) ili dešifrovati (d) primenom Trifid algoritma. Od drugog reda na dalje nalazi se tekst koji je potrebno obraditi. Kao rezultat izvršavanja programa, potrebno je u izlazni fajl, prema formatu specificiranom za ulazni fajl upisati mode, koji je suprotan od onog koji je definisan u ulaznom fajlu i rezultat obrade.

.....

Ovaj algoritam prilikom šifrovanja i dešifrovanja koristi 3 matrice karaktera dimenzija 3×3:

	Layer 1			Layer 2			Layer 3		
#	1	2	3	1	2	3	1	2	3
1	A	B	V	J	K	L	S	T	U
2	D	E	F	M	N	O	V	W	X
3	G	H	I	P	Q	R	Y	Z	.

Postupak šifrovanja najlakše je objasniti na primeru:

Poruka koja se šifruje: r a c u n a r s k a e l e k t r o n i k a .

Layer: 2 1 1 3 2 1 2 3 2 1 1 2 1 2 3 2 2 2 1 2 1 3

Kolona: 3 1 3 3 2 1 3 1 2 1 2 3 2 2 2 3 3 2 3 2 1 3

Red: 3 1 1 1 2 1 3 1 1 1 2 1 2 1 1 3 2 2 3 1 1 3

Sledeći korak je formiranje trojki iščitavanjem redova sleva na desno:

211 321 232 112 123 222 121 331 332 131 212 322 233 232 133 111 213 111 212 113

223 113

a zatim se ovde trojke koduju ponovo korišćenjem istih matrica, tako što prvi broj predstavlja layer, drugi kolonu, a treći red, tako da je dobijena šifrovana poruka:

jtodhnbuxcmwroiapamggg.

---

# KONCEPT:

---

Kod programa je podeljen na 3 smislene celine:

- main proc
- encryption proc
- decryption proc

Pored ove 3 celine na početku su ispisani *makro*-i koji su olakšali implementiranje i preglednost programa, kao i niske koje služe kao interfejs i koje su sklone promenama da bi zadovoljili potrebe korisnika. Za implementaciju rešenja problema korišćene su 2 biblioteke:

- Irvine32.inc
- Macros.inc

Irvine32 zbog lakše korišćenja ispisa i upisa u datoteke i komandnu liniju, uspostavljanja interfejsa za korisnika, dok je Macros.inc kao što smo ranije napomenuli zbog preglednosti koda.

Takođe pre koda programa su definisane konstante i promenljive, kao i niske koje ćemo kasnije koristiti za interfejs.

## Main procedura

Main procedura je zadužena za interfejs i dizanje *exception*-a u slučaju greške korisnika, ili sistemske greške u otvaranju datoteke. Zapisuje imena ulazne i izlazne datoteke, ulazna datoteka čita i upisuje u bafer (*buffer*) u kome se tekst obrađuje pre nego što se upisuje u izlaznu datoteku. U zavisnosti od prvog karaktera u tekstu ulazne datoteke, pokreće se procedura šifrovanja ili dešifrovanja ulaznog teksta. Unutar main procedure vrši se i zatvaranje datoteka u slučaju da su uspešno otvoreni.

## Encryption procedura

Encryption procedura ili procedura šifrovanja se pokreće u slučaju da je tako nagovešteno u prvom karakteru ulazne datoteke (e). Koristi se tabela iz postavke zadatka i prave se 3 niza brojeva (koji mogu biti 1 2 3 kao što je dato u postavci), i od ta 3 niza formiramo jedan niz jednostavnim nadovezivanjem jednog na drugi, tako što svaku trojku pomoću datih tabela menjamo karakterom. Sa promenjenom niskom nazvanom *buffer* vraćamo se main proceduri.

## Decryption procedura

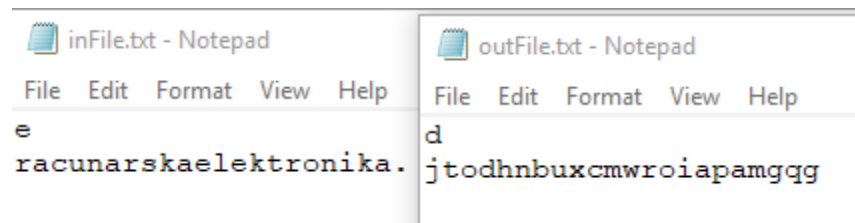
Decryption procedura ili procedura šifrovanja predstavlja postupak suprotan od onog unutar decryption procedure.

---

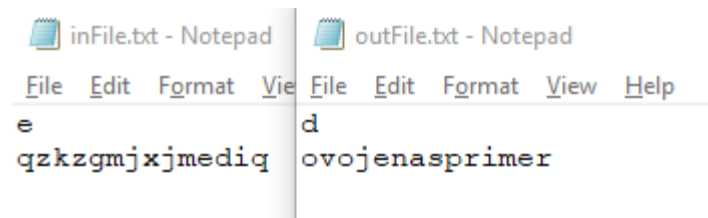
# TEST PRIMERI:

---

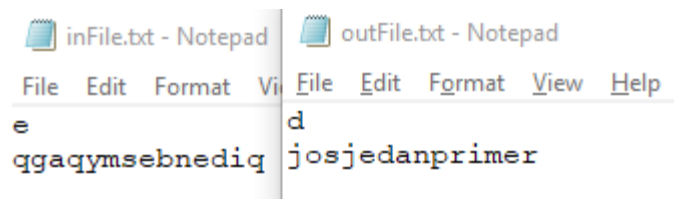
Primer dat u postavci zadatka:



Ovo je jedan naš primer:



I još jedan primer:

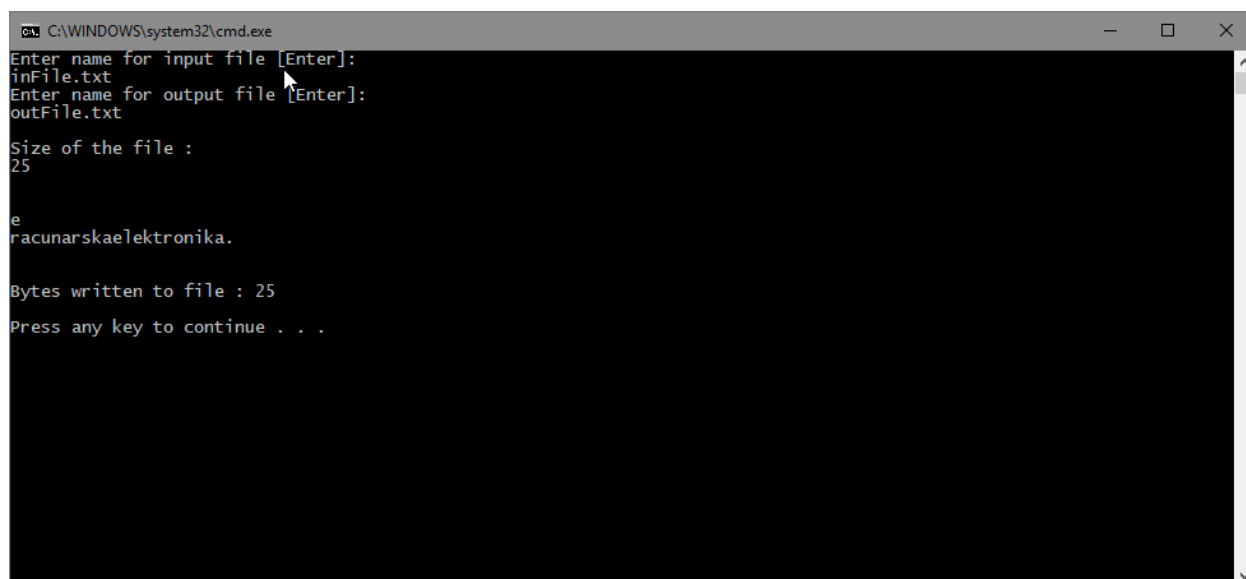


---

# INTERFEJS U KOMANDNOJ LINIJI:

---

Program je interaktivan sa korisnikom i zahteva od njega da unese naziv ulazne i izlazne datoteke. Ono što izbacuje jeste sadržaj ulazne datoteke u komandnoj liniji kako bi se korisnik uverio da je to tekst koji je želeo da šifrjuje kao i broj bajtova koji su korišćeni za pisanje nove datoteke i oni koji sadrže tekst iz ulazne datoteke.



```
C:\WINDOWS\system32\cmd.exe
Enter name for input file [Enter]:
inFile.txt
Enter name for output file [Enter]:
outFile.txt

Size of the file :
25

e
racunarskaelektronika.

Bytes written to file : 25
Press any key to continue . . .
```

U slučaju da korisnik unese naziv za izlaznu datoteku koji je već korišćen kao na primer ulaznu datoteku, neće moći da napravi i prijavice grešku pre bilo kakvog šifrovanja ili dešifrovanja:

```
C:\WINDOWS\system32\cmd.exe
Enter name for input file [Enter]:
inFile.txt
Enter name for output file [Enter]:
inFile.txt

Size of the file :
25

e
racunarskaelektronika.

Cannot create file
Press any key to continue . . .
```

Ako pokušamo da otvorimo datoteku za čitanju koja ne postoji prijavio se sistemski greška:

```
C:\WINDOWS\system32\cmd.exe
Enter name for output file [Enter]:
sdkjfl.txt
Enter name for input file [Enter]:
tsssgjd.txt
Error 2: The system cannot find the file specified.
Press any key to continue . . .
```

U kodu su obrađene još neke greške koje se dešavaju ređe pa nisu predstavljanje u izveštaju, npr. sistemski greške ili prekoračenje u dužini imena datoteke.

---

# NAPOMENE:

---

Korišćene su procedure i kod iz biblioteke Kipa Irvina, kao i delovi koda iz materijala sa sajta predmeta. Kod otvaranja datoteka se nije vodilo računa o pravima pristupa, tj. iako nemamo nameru da menjamo ulaznu datoteku, ona se otvara sa svim pravima (nije testirano na datotekama na kojima korsnik ima samo pravo čitanja). Takođe maksimalna dužina naziva datoteke je data u kodu programa kao 80 karaktera, što se može podesiti, međutim dužina naziva datoteke može da bude ograničena operativim sistemom (npr. kod Windows-a dužina celog *path*-a može biti najviše 256 karaktera, tako da zavisi od toga gde se nalaze datoteke ...). Program nije najpreglednije rešen jer umesto velikog broja makroa je bilo moguće koristii niz sa elementom kao kodom i tako iterirati kroz niz ali je napravljen kompromis i ti makroi su smešteni u niz koji se koristi kasnije u programu. Takođe mogli smo da da kodiranje i dekodiranje da izvedemo kao procedure ali ono što bi dobili time ne bi bilo vredno uloženog truda.

---

# KOD PROGRAMA:

---

```
;List of libraries

include Irvine32.inc
include Macros.inc

;List of macros

;.model flat,stdcall alewady included

endl equ <0dh,0ah> ; this is a macro

ascii_d equ <64h>
ascii_e equ <65h>
ascii_fullStop equ <2eh>

; it goes like layer row column
coded_a equ <010101b>
coded_b equ <011001b>
coded_c equ <011101b>
```



```

coded_d equ <010110b>
coded_e equ <011010b>
coded_f equ <011110b>
coded_g equ <010111b>
coded_h equ <011011b>
coded_i equ <011111b>
coded_j equ <100101b>
coded_k equ <101001b>
coded_l equ <101101b>
coded_m equ <100110b>
coded_n equ <101010b>
coded_o equ <101110b>
coded_p equ <100111b>
coded_q equ <101011b>
coded_r equ <101111b>
coded_s equ <110101b>
coded_t equ <111001b>
coded_u equ <111101b>
coded_v equ <110110b>
coded_w equ <111010b>
coded_x equ <111110b>
coded_y equ <110111b>
coded_z equ <111011b>
coded_fullStop equ <111111b>

; let's make buffer in buffer_size

BUFFER_SIZE = 501    ; this is max_size
TRIPLE_BUFFER_SIZE = BUFFER_SIZE*3
BUFFER_SIZE_PLUS_THREE = BUFFER_SIZE+3

.data?
inFileHandle handle ?
outFileHandle handle ?
buffer byte BUFFER_SIZE dup(?)
inputFilename byte 80 dup(?) ; input txt file must be less than 80 characters long
outputFilename byte 80 dup(?)
bytesToEnc dword ?
coded byte TRIPLE_BUFFER_SIZE dup (?)
code_to_output byte BUFFER_SIZE_PLUS_THREE dup (?)
output_string_len dword ?
bytesWritten dword ?

.data
;outputFilename byte "outFile.txt",0 ; end of string
error_outfile byte "Cannot create file",endl,0 ; line feed carriage return == CR
error_infile byte "File not opened",endl,0
bytesWrittenstring byte "Bytes written to file : ",0
outputInterface byte "Enter name for output file [Enter]: ",endl,0
inputInterface byte "Enter name for input file [Enter]: ",endl,0
readError byte "Error reading a file ",endl,0
smallBuffer byte "Buffer too small for the input file",endl,0
fileSize byte endl,"Size of the file : ", endl,0
doubleCR byte endl,endl,0
alphabet_iterator byte 60h

```

```

coded_alphas byte coded_a, coded_b, coded_c, coded_d, coded_e, coded_f, coded_g, coded_h,
coded_i, coded_j, coded_k, coded_l, coded_m, coded_n, coded_o, coded_p, coded_q, coded_r,
coded_s, coded_t, coded_u, coded_v, coded_w, coded_x, coded_y, coded_z
coded_fs byte coded_fullStop

.code

decryption proc c uses esi edi ecx edx,
               num_char_bytes:dword, iterator_decryption:dword

    mov eax,iterator_decryption
    mov eax,offset code_to_output
    mov eax,offset coded

    mov ecx,num_char_bytes
    xor esi,esi    ; mov esi,0
    xor edi,edi

code_letters:
    xor eax,eax
    mov esi,iterator_decryption
    mov al,[esi]
    xor esi,esi
    cmp al,ascii_fullStop
    je full_stop_label_code

iterate_through_alphabet:
    inc esi
    inc alphabet_iterator
    cmp al,alphabet_iterator
    jne iterate_through_alphabet

    mov alphabet_iterator,60h ;resetting it for the next iteration
    mov al,coded_alphas[esi-1]

    jmp letter_found

full_stop_label_code:
    mov al,coded_fs
    ;independent from the alphabet

letter_found:

    mov dl,al
    and al,30h
    shr al,4
    mov coded[edi],al
    inc edi
    mov al,dl
    and al,0ch
    shr al,2
    mov coded[edi],al
    inc edi
    mov al,dl
    and al,03h
    mov coded[edi],al

```

```

        inc edi
        inc iterator_decryption
        ; we are going to iterate through it just once
        loop code_letters
        ;and of big loop

        xor esi,esi
        mov ecx,num_char_bytes
decode_loop:

        xor eax,eax
        xor edx,edx
        mov al,coded[esi]
        shl al,4
        or dl,al
        add esi,num_char_bytes
        mov al,coded[esi]
        shl al,2
        or dl,al
        add esi,num_char_bytes
        mov al, coded[esi]
        sub esi,num_char_bytes
        sub esi,num_char_bytes
        or dl,al
        mov al,dl
        ;coded

        mov edi,-1
        cmp al,coded_fullStop
        je full_stop_label_decode
decode_alpha:
        inc edi          ; here we change
        cmp al,coded_alphas[edi]
        jne decode_alpha
        xor eax,eax
        mov al,61h
        add eax,edi
        jmp skip_full_stop
full_stop_label_decode:
        mov al,ascii_fullStop

skip_full_stop:
        mov code_to_output[esi+3],al

        inc esi
        loop decode_loop

        ret
decryption endp

encryption proc c uses esi edi ecx edx,
                    num_char_bytes:dword, iterator_encryption:dword
        mov eax,offset code_to_output
        mov eax,offset coded
        xor esi,esi
        xor edi,edi
        mov ecx,num_char_bytes

```

```

code_encryption:
    xor eax,eax
    mov esi,iterator_encryption
    inc iterator_encryption
    mov al, [esi]
    xor esi,esi
    cmp al,ascii_fullStop      ; it's not similar to the ascii for the alphas, so
special treatment
    je fullStoplabel

iterate_through_alphabet:
    inc esi      ; here we change
    inc alphabet_iterator
    cmp al,alphabet_iterator
    jne iterate_through_alphabet
    mov alphabet_iterator,60h ;reset the iterator on -1 state
    mov al,coded_alphas[esi-1]
    jmp skip_full_stop_code

fullStopLabel:
    mov al,coded_fs

skip_full_stop_code:

    mov dl,al
    and al,30h
    shr al,4
    mov coded[edi],al
    add edi,num_char_bytes
    mov al,dl
    and al,0ch
    shr al,2
    mov coded[edi],al
    add edi,num_char_bytes
    mov al,dl
    and al, 03h
    mov coded[edi],al
    sub edi,num_char_bytes
    sub edi,num_char_bytes

    inc edi
    loop code_encryption

    xor esi,esi
    xor edi,edi
    mov ecx,num_char_bytes
decode_loop:
    xor eax,eax
    xor edx,edx
    mov al, coded[esi]
    shl al,4
    or dl,al
    inc esi
    mov al,coded[esi]
    shl al,2
    or dl,al
    inc esi
    mov al,coded[esi]

```

```

        inc esi
        or dl,al
        mov al,dl
        push esi
        mov esi,-1
        cmp al,coded_fullStop
        je full_stop_decode
decode_alphas:
        inc esi          ; here we change
        cmp al,coded_alphas[esi]
        jne decode_alphas
        xor eax,eax
        mov al,61h       ; ascii code for a
        add eax,esi
        jmp skip_full_stop_decode
full_stop_decode:
        mov al,ascii_fullStop

skip_full_stop_decode:
        pop esi
        mov code_to_output[edi+3],al
        inc edi
        ;add iterator_encryption,1
        loop decode_loop; ecx should decrement

        ; code to output is going to be used, we left 3 bytes at the start purposefully...
        ret
encryption endp

main proc

        mov edx, offset outputInterface
        call WriteString
        mov edx, offset outputFilename
        mov ecx, sizeof outputFilename
        call ReadString

        mov edx, offset inputInterface
        call WriteString
        mov edx, offset inputFilename
        mov ecx, sizeof inputFilename ; sizeof and lengthof should be the same if the
quantity is a byte
        call ReadString

        mov edx, offset inputFilename
        call OpenInputFile
        mov inFileHandle, eax

        ; check if there was an error opening the file
        cmp eax,INVALID_HANDLE_VALUE
        jne no_error1
        mov eax, offset readError
        ;call WriteString
        call WriteWindowsMsg
        jmp quit

no_error1:

```

```

    mov edx, offset buffer ; where should it be written
    mov ecx, BUFFER_SIZE
    call ReadFromFile      ; this is the actual READ CALL
    jnc check_buffer_size ; checks how? if the carry flag is zero
    mov eax, offset readError
    call WriteString
    call WriteWindowsMsg
    jmp close_file

check_buffer_size:
    push eax
    sub eax, 3
    mov bytesToEnc, eax
    pop eax
    cmp eax, BUFFER_SIZE
    jb buffSizeOK ; guess BUFFER_SIZE > eax than jump
    mov edx, offset smallBuffer
    call WriteString
    jmp quit ; this shall be a label at the end of the file

buffSizeOK:
    mov buffer[edx], 0 ; string terminator
    mov edx, offset fileSize
    call WriteString
    call WriteDec
    call Crlf
    mov edx, offset doubleCR
    call WriteString
    mov edx, offset buffer
    call WriteString
    call Crlf
    mov edx, offset doubleCR
    call WriteString

    mov eax, offset buffer
    add eax, 3
    push eax
    push bytesToEnc

    xor eax, eax
    mov al, byte ptr buffer[0]
    cmp al, ascii_e
    je encrypt
    cmp al, ascii_d
    je decrypt
    jmp close_file

decrypt:

    call decryption
    mov code_to_output, ascii_e
    jmp skip_encryption

encrypt:

    call encryption
    mov code_to_output, ascii_d

```

```

skip_encryption:

    mov esi,1
    mov code_to_output[esi], 0dh
    inc esi
    mov code_to_output[esi], 0ah

    mov edx, offset outputFilename
    call CreateOutputFile
    mov outFileHandle,eax
    cmp eax,INVALID_HANDLE_VALUE
    jne file_ok
    mov edx, offset error_outfile
    call WriteString
    jmp quit

file_ok:
    mov eax,bytesToEnc
    add eax,3
    mov output_string_len,eax

    ;Write the buffer to the output file.
    mov eax,outFileHandle
    mov edx,OFFSET code_to_output
    mov ecx,output_string_len
    call WriteToFile
    mov bytesWritten,eax ; save return value
    call CloseFile

    ; Display the return value.
    mov edx,offset bytesWrittenstring ; "Bytes written"
    call WriteString
    mov eax,bytesWritten
    call WriteDec
    call Crlf
    call Crlf

close_file:
    mov eax,inFileHandle
    call CloseFile

quit:
    invoke ExitProcess,0

main endp
end main

```